optim_functions

These functions are for the optimization process.

Model parameters appearing here are better described in CMV_Models.pdf documentation. The optimization procedure proceeds as follows:

1. a model (passed as a function) and parameters (fixed and to be optimized) for the model are passed into the nloptr wrapper (nloptr_call) directly from a specific model fitting script (they are randomly drawn using LHS). Parameter constraints are applied to the solver call if they are given.
2. nloptr_call calls nloptr::nloptr with appropriate settings which optimizes optimize_fun and passes in the model function and parameters (fixed and to be optimized).
3. If the parameters result in a very large R0, then the model simulation fails. So there is a check for this that returns an mse = Inf. I think this might be legacy code because the LHS code accounts for this by limiting the range of beta.
4. The model is simulated with desolve::lsoda.
5. Oscillatory results (highly periodic simulations that go through every day point) are heavily penalized by given mse of Inf.
6. optimize_fun calls cost_function passing model simulation results and infant episode data. Model simulation results are compared to viral data at common points using least squares. least squares calculation is returned.

Initial value note: This version is written with intention that $I_0 = 1$ (initial infected cells with replicating virus) and the start_day (before data) is fitted. start_day works by having the model start simulation at -start_day, so that day 0 in the model is day 0 in the data (first positive). For other initial values, $S_0 = \mathrm{K}$, $I0_0 = 0$ (the latently infected compartment, would be L_0 in manuscript), V = 0, and Tcell = 0 (E in the manuscript).

The initial virus settings are from legacy code and it is not advised to fit the initial viral load (before detection) at a fixed start time. AUC_data is also legacy code for a different implementation of immune response. It is effectively equivalent to setting γ (called 'death' in CMV_Models.R) = 0 (see CMV_models.pdf).

Functions

1. nloptr_call

wrapper for nloptr. Has two separate calls depending on if bounds are specified. Also post-processing of output include de-logging fitted values and returning a simple data.frame that may (justParms = F) or may not (justParms = T) include fit information.

```
### This script was created by model_code_documentation/create_scripts.sh
### The following functions are described in model_code_documentation/optim_functions.Rmd with correspon

nloptr_call = function(fitted_parms, fixed_parms, fit_data, model_initial, CMV_model = CMVModel_latent_I
                       lowerBounds = NULL, upperBounds = NULL,  R0_test = T, AUCData = NULL,
                       justParms = F, maxeval = 1000){
  #calls the base model by default

  #if either set of constraints are missing, runs unconstrained
  if(is.null(upperBounds) | is.null(lowerBounds)){
    fit = nloptr(
```

```
      x0 = fitted_parms,
      eval_f = optimize_fun,
      opts = list("algorithm"="NLOPT_LN_NELDERMEAD",
                  "xtol_rel"=1.0e-8, "maxeval" = maxeval),
      model_fun = CMV_model, inData = fit_data,
      inParms = fixed_parms,  init = model_initial, AUCData = AUCData,
      parmNames = names(fitted_parms), R0_test = R0_test)
  }
  else{
    fit = nloptr(
      x0 = fitted_parms,
      eval_f = optimize_fun,
      lb = unname(lowerBounds),
      ub = unname(upperBounds),
      opts = list("algorithm"="NLOPT_LN_NELDERMEAD",
                  "xtol_rel"=1.0e-8, "maxeval" = maxeval),
      model_fun = CMV_model, inData = fit_data,
      inParms = fixed_parms,  init = model_initial, AUCData = AUCData,
      parmNames = names(fitted_parms), R0_test = R0_test)
  }
  parmsfit = 10^fit$sol
  names(parmsfit) = names(fitted_parms)

  outparms = c(parmsfit, fixed_parms)

  fit_output = as.data.frame(t(outparms))

  #default to include additional fit information
  if(!justParms){
    fit_output$mse = fit$obj
    fit_output$conv = fit$status
  }

  fit_output
}
```

2. optimize_fun

called by nloptr for optimizing the fitparms. Can be called independently. R0_test uses crude test of R0 as a boundary for bad parameters to avoid weird fits (R0 < 88). This also checks for highly oscillatory fits by penalizing models that peak within 5 days.

```
optimize_fun = function(fitParms, inParms, inData, init, model_fun = CMVModel_latent_linear, parmNames =
                        AUCData = NULL, R0_test = F){
  inData = arrange(inData, days2)

  fitParms = 10 ^ fitParms
  if(length(parmNames) != 0) names(fitParms) = parmNames

  id_var = which(names(inParms) == "id")

  parms = c(fitParms, inParms)
```

```
  init["S"] = as.numeric(unname(parms["K"]))
  init["I"] = as.numeric(unname(parms["initI"]))
  init["V"] = as.numeric(unname(parms["initV"]))

  start_time = -round(unname(parms["start_day"]), 1)
  times = seq(start_time, max(inData$days2) + 50, 0.1)

  # this throws out parameters that have R0 > 100, this is the old R0 and is missing (1+mu/alpha) in de
  # so the actual constraint is > 82 since mu = 1/4.5 and alpha =1, that term =1.22
  # so when this R0 equals 100, the actual R0 is 100/1.22 - 81.97

  if(R0_test) if(with(as.list(c(fitParms, inParms)), (beta * K * p /(delta * c))) > 100) return(Inf)

  model_out = as.data.frame(lsoda(init, times, model_fun, parms, AUCData = AUCData))

  if (sum(model_out$V) == "NaN" | dim(model_out)[1] != length(times)) return(Inf)

  #if there is early oscillation
  if(model_out$time[which.max(model_out$V)] < 5) return(Inf)

  mse = cost_function(model_out, inData)
  if (mse == 0) return(Inf) # browser()
  return(mse)
}
```

2b. optimize_fun_test

Use this with results to calculate mse (generally for troubleshooting)

```
optimize_fun_test = function(fitOutput, inData, AUCData = NULL){

  modelIn = fitOutput$model

  if(modelIn == "CMVModel_latent_linear" | is.null(fitOutput$theta)) {
    init = with(fitOutput, c(S = K, I0 = 0, I = initI, V = initV))
    }else init = with(fitOutput, c(S = K, I0 = 0, I = initI, V = initV, Tcell = 0))

  times = seq(-round(fitOutput$start_day, 1), max(inData$days2) + 50, 0.1)

  out = as.data.frame(lsoda(init, times, get(modelIn),
                            unlist(select(fitOutput, -model, -PatientID)), AUCData = AUCData))

  inData = arrange(inData, days2)

  mse = cost_function(out, inData)
  #print(mse)
  if (mse == 0) return(Inf) # browser()
  return(mse)
}
```

3. cost_function

mse function using simulated model data and raw data. Called by optimize_fun but can be called independently.

```
cost_function = function(model, data, debug = F){
  if(debug) browser()
  data = arrange(data, days2)
  data = data %>% dplyr::mutate(days_model = days2)

  sample_model = arrange(model[which(round(model$time, 1) %in% round(data$days_model, 1)), ], time)
  mse = sum((log10(sample_model$V) - data$count2)^2, na.rm = T)
  if (mse == "NaN") browser()
  #print(mse)
  return(mse)
  }
```